Investigating Applications Portability with the Uintah DAG-based Runtime System on PetaScale Supercomputers

Qingyu Meng
Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, UT 84112 USA
qymeng@sci.utah.edu

Alan Humphrey
Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, UT 84112 USA
ahumphrey@sci.utah.edu

John Schmidt
Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, UT 84112 USA
jas@sci.utah.edu

Martin Berzins
Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, UT 84112 USA
mb@sci.utah.edu

ABSTRACT

Present trends in high performance computing present formidable challenges for applications code using multicore nodes possibly with accelerators and/or co-processors and reduced memory while still attaining scalability. Software frameworks that execute machine-independent applications code using a runtime system that shields users from architectural complexities offer a possible solution. The Uintah framework for example, solves a broad class of large-scale problems on structured adaptive grids using fluid-flow solvers coupled with particle-based solids methods. Uintah executes directed acyclic graphs of computational tasks with a scalable asynchronous and dynamic runtime system for CPU cores and/or accelerators/co-processors on a node. Uintah's clear separation between application and runtime code has led to scalability increases of 1000x without significant changes to application code. This methodology is tested on three leading Top500 machines; OLCF Titan, TACC Stampede and ALCF Mira using three diverse and challenging applications problems. This investigation of scalability with regard to the different processors and communications performance leads to the overall conclusion that the adaptive DAG-based approach provides a very powerful abstraction for solving challenging multi-scale multi-physics engineering problems on some of the largest and most powerful computers available today.

Categories and Subject Descriptors

D.1.3 [**Software**]: Concurrent Programming; G.1.8 [**Mathematics of Computing**]: Partial Differential Equations; G.4 [**Mathematics of Computing**]: Mathematical Software; J.2 [**Computer Applications**]: Physical Sciences and Engineering

General Terms

Design, Performance, Algorithms, Application, Software

Keywords

Uintah, hybrid parallelism, scalability, parallel, adaptive, Xeon Phi, Blue Gene/Q, heterogeneous systems, GPU, co-processor

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SC '13, November 17-21, 2013, Denver, Colorado, USA.
Copyright 2013 ACM 978-1-4503-2378-9/13/11 ...$15.00.

1. INTRODUCTION

The trend towards larger and more diverse computer architectures with or without co-processors and GPU accelerators and differing communications networks poses considerable challenges for achieving both performance and scalability when using general purpose parallel software for solving multi-scale and multi-physics applications. One approach that is suggested as a suitable candidate for exascale problems is to use directed acyclic graph (DAG) based codes, [2]. Such codes have the advantage that the tasks in the task-graph may be executed in an adaptive manner and if enough tasks are available, choosing an alternate task may avoid communications delays. At the same time if the applications software is written in an abstract task-graph manner so as to be

executed by a runtime system that adaptively varies task-graph execution, then it may be possible to combine a modest degree of portability across a number of large-scale parallel architectures with scalability at large core counts.

The aim in this paper is to illustrate that this combination of portability and scalability can be demonstrated with the Uintah software `www.uintah.utah.edu` on three of the seven fastest computers as measured by the top 500 list of November 2012 [62], DOE's Titan and Mira and NSF's Stampede, [15,43,60]. These machines make uses of three very different processors and networks. Two of the machines, Titan and Stampede have GPU accelerators and Intel Xeon Phi co-processors respectively. The approach used here will be to take three representative and challenging Uintah applications codes and to examine their scalability and performance on these very different machines. The three applications are:

- Fluid-structure interaction with adaptive mesh refinement in an example used by [42] on the DOE Titan system;
- Radiation modeling through raytracing with significant amounts of global communication as used by [25] on a variety of GPU accelerators and CPUs; and
- Turbulent combustion on a fixed-mesh requiring large-scale linear solves with the Hypre iterative solver [51] again used on the DOE Titan system.

These three applications have very different communications requirements and work patterns. The approach used here will be to run these three applications at large scale on each of the target machines and to demonstrate how the Uintah runtime manages to achieve scalability through adaptive execution of its task graph and in the context of three different communications patterns for each of the three applications. This scalability will be shown as follows. In Section 2 the Uintah software will be outlined with the main components of the runtime system. The salient features of three target architectures and the porting of the Uintah code from the Titan architecture to the other two architectures are described in Section 3. Section 4 will describe the simulation components of the Uintah used to solve problems and analysis of the components' example communications and task execution patterns on the three machines. In Section 5, the scalability and performance results obtained will be given with an analysis of the different cases to show how the task-graph execution pattern adaptively varies to achieve scalability. Section 6 will describe related work on task-graph based approaches and other frameworks. Overall we will show that with these applications we are able to achieve good scalability on machines like Mira and Stampede when coming from our starting point of scalability on Titan [42]. Moreover such scalability comes without significant porting effort. The paper concludes with a discussion of how these results may be viewed as representative and how they may or may not extend to future architectures with even greater use of accelerators and co-processors. Overall our conclusion is that the adaptive DAG-based approach provides a very powerful abstraction for solving challenging multi-scale multi-physics engineering problems on some of the largest and most powerful computers available today.

2. UINTAH INFRASTRUCTURE

The Uintah software framework originated in the University of Utah DOE Center for the Simulation of Accidental Fires and Explosions (C-SAFE) [14] (9/97-3/08) which focused on providing software for the numerical modeling and simulation of accidental fires and explosions. The Uintah open-source (MIT License) software has been used to solve many different challenging fluid, solid and fluid-structure interaction problems. The present status of Uintah, including applications, is described by [4].

Uintah's parallel software components facilitate the solution of partial differential equations (PDEs) on structured adaptive mesh refinement (SAMR) grids. Uintah makes it possible to integrate multiple simulation components, analyze the dependencies and communication patterns between these components, and efficiently execute the resulting multi-physics simulation. Uintah contains four main simulation components: 1) the ICE [31,32] code for both low and high-speed compressible flows; 2) the multi-material particle-based code MPM [56] for structural mechanics; 3) the combined fluid-structure interaction (FSI) algorithm MPM-ICE [21,22]; and 4) the ARCHES turbulent reacting CFD component [27, 53] that was designed for simulation of turbulent reacting flows with participating media radiation. Uintah scales well on a variety of machines at small to medium scale (typically Intel or AMD processors with Infiniband interconnects) and on larger Cray machines such as Kraken and Titan as is shown by [6, 35]. Uintah also runs on many other NSF and DOE parallel computers (Stampede, Keeneland, Mira, etc). Furthermore Uintah is also used by many DOD, DOE and NSF research projects in areas such as angiogenesis, tissue engineering, green urban modeling, blast-wave simulation, semi-conductor design and multi-scale materials research [4].

The component-oriented approach that Uintah is based upon [45, 46] allows the application developer to only be concerned with solving the partial differential equations on a local set of block-structured adaptive meshes, without worrying about explicit message passing calls in MPI, or indeed parallelization in general. This is possible as the parallel execution of the tasks is handled by a runtime system that is application-independent. This division of labor between the application code and the runtime system allows the developers of the underlying parallel infrastructure to focus on scalability concerns such as load balancing, task (component) scheduling, communications, including accelerator or co-processor interaction. This component-oriented parallel programming approach also makes it possible to leverage advances in the runtime system, allowing improvements in scalability to be immediately applied to applications without any additional work by the application developer. This type of programming model is ideally suited for a software environment like Uintah and has contributed significantly to its scaling success. Nevertheless, the applications developer must still write code that does not use excessive communication in relation to computation and/or a large number of global communications operations. Should this unfortunate situation occur, Uintah's detailed monitoring system is often able to identify the source of the inefficiency.

Uintah components are C++ classes that follow a simple interface to establish connections with other components in the system. Uintah utilizes a task-graph of parallel computation and communication to express data dependencies between multiple application components. The task-graph is a directed acyclic graph (DAG) in which each task reads inputs from the preceding task and produces outputs for the subsequent tasks. the same DAG per patch. A Uintah patch is a hexahedral cube of grid cells and the smallest data parallel unit. The task's inputs and outputs are specified for a generic patch in a structured SAMR grid, thus a DAG will be created with tasks of only local patches. Each task has a C++ method for the actual computation and each component specifies a list of tasks to be performed and the data dependencies between them [5].

The methodological approach behind Uintah is described in detail in [14, 45, 46] and builds upon the innovative original task-graph design of Parker [45], but with a runtime system, data structures and load balancers that have been rewritten to ensure much-improved scalability. The general task-graph approach of Uintah is similar to that of Charm++ [28], but makes use of its own unique architecture and algorithms. For example, Uintah uses a "data warehouse" through which all data transfer takes place and which ensures that application codes are isolated from the MPI communications layer. This degree of separation between the runtime system of Uintah and the applications codes written using a component-oriented parallel programming model makes it possible to achieve great increases in scalability through changes to the runtime system that executes the taskgraph, *without changes to the applications components themselves*.Particular advances made in Uintah are scalable adaptive mesh refinement [36–38] coupled to challenging multiphysics problems. [5, 35] and a load balancing data assimilation and feedback approach [35], which out-performs traditional cost models. A key factor in improving performance is the reduction in wait time through the dynamic and even out-of-order

execution of tasks [5,41] in the Uintah runtime system. The need to reduce memory use in Uintah led to the adoption of a nodal shared memory model in which there is only one MPI process per multicore node, one global memory per node and execution on individual cores is through threads bound to available cores on-node. This has made it possible to reduce memory use by a factor of 10 and to increase the scalability of Uintah to 256K cores on complex fluid-structure interactions with adaptive mesh refinement [42] on DOE's Titan and up to 512K cores on DOE's Mira. Uintah's thread-based runtime system as shown in Figure 1 uses:
(1) Decentralized execution [41] of the task-graph is implemented by each CPU core requesting work itself. This eliminates having a single controlling thread on a core and possible under use or contention. (2) A Shared Memory Abstraction through Uintah's data warehouse hides message passing from the user but at the cost of multiple cores accessing the warehouse. This shared memory [41] approach is implemented by making use of atomic lock-free operations (supported by modern CPUs) that allow efficient access by all cores to the shared data on a node. (3) Accelerator task execution [25] on a node uses a GPU Data Warehouse system that enables tasks to be executed efficiently through preloading of data on multiple accelerators per node, as on NSF's Keeneland system [25]. The key to the success of Uintah is the process of generating the task-graph and then executing it in as efficient as possible a manner.

2.1 Task Graph Generation

Uintah uses a grid of hexahedral cells defined in the Uintah input file. This Uintah grid can contain one or more levels with different resolutions while each level is further divided into smaller hexahedral patches. When running with SAMR, finer grid levels are created by the Uintah Regridder [36]. Since finer grid levels may not be continuous across the domain, Uintah uses a binary bounding volume hierarchy (BVH) tree to save patches on a particular grid level. After patches on each grid level are created, the Uintah load balancer is used to partition and assigned patches to MPI nodes. By profiling execution time on each patch, this load balancer can use history data to predict the further work load [35]. Once each multicore node has its patches assigned, tasks are then created on patches. A Uintah task is defined by three major attributes: 1) the "require" variables it needs to start computing with the number of ghost cells; 2) the "compute" variables that it will calculate; and 3) A serial call back function that can run on any patch. In this way, the user only needs to write serial code on a generic patch. When the call back happens, the actual patch is then fed into the task.

Uintah uses a process of checking the overlap of the input variables and the output variables for each task that makes it possible to create a directed acyclic graph by connecting the input and output of tasks. During the task graph compilation process, the correctness of the task graph is checked. Errors such as missing or ambitious dependencies (requires without computes, double computes) or cyclic dependencies (i.e. two or more task depend on each other) will be reported to simulation component developer. This reporting helps developers to write correct and consistent task inputs and outputs. In the case when a dependency that connects two tasks associated with patches on the same multicore node is found, it is tagged as an internal dependency. Similarly in the case when a dependency that connects two tasks associated with patches on different multicore nodes is found, an external dependency is created. At the end of task graph compilation an MPI message tag is assigned to each external dependency. As Uintah only creates tasks on local and neighbouring patches, task graph compilation can be done in parallel without any communications between multicore nodes. Once the task graph is compiled, it can be used for multiple timesteps without being recompiled unless the grid changes such as when new patches are created or deleted by the SAMR regridder or when the load balancer moves patches from one node to another.

2.2 Task Graph Execution

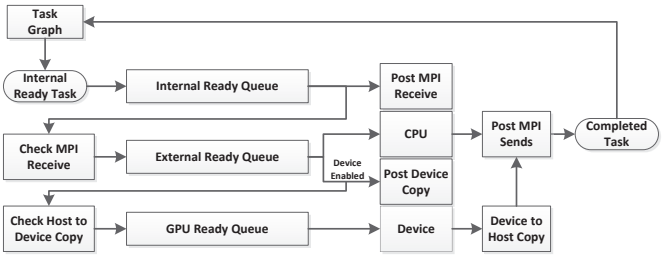

Figure 1: Uintah Multi-Queue Task Scheduling

Efficient execution of the Uintah Task Graph is achieved through the Uintah runtime system which has three important features that help to support task execution. These three features are: 1) variable memory management though Uintah's data warehouse; 2) task management through a multi-queue scheduler; and 3) MPI and device communication management.

The Uintah data warehouse is a dictionary that maps a variable's name and patch ID to its memory location. Every Uintah task uses the data warehouse to allocate variables, read variables and update variables. This approach also enables the data warehouse to track the life span of any variable. Variable pointers that existed on a previous timestep are stored in the read-only *old data warehouse* and variables pointers that are computed or updated during the current timestep are stored in a *new data warehouse*. At the end of each timestep, the variables in the new data warehouse are then mapped to the old data warehouse for the next timestep in the simulation and a new data warehouse is initialized. When a variable is no longer being used by any task, the data warehouse will automatically delete this variable and free the corresponding memory. The data warehouse uses lock-free data structures to allow tasks on multiple threads to correctly access the data at the same time without being serialized [42].

The Uintah scheduler uses multiple queues to track the status of each task. As shown in Figure 1, when all of a task's internal dependencies are satisfied this task will be placed into the internal ready queue. When all of a task's MPI messages are received, the task is then moved to the external ready queue, and ultimately to a device ready queue if this task is device (accelerator or co-processor) enabled. All of a device-enabled task's data are then copied to that device. This multi-stage queue design [42] allows multicores or accelerators/co-processors to post MPI Isend/Irecieve, prepare data to copy from/to a device, run CPU tasks or run device tasks all at the same time dynamically. Finally, when a task is finished, the next internal task is identified from the task graph and the process is repeated. While this level of description provides the necessary background it is also important to note that the task queues are priority queues and tasks may be executed out-of-order to improve efficiency [41].

2.3 GPU and Co-Processor Task Management

Finally the nodal architecture of Uintah has been extended to run tasks on one or more accelerators or co-processors [25, 39]. As shown in Figure 1, a key design feature of the Uintah task scheduler is its multi-stage queue architecture for efficient scheduling of both CPU and accelerator/co-processor tasks. For efficient GPU task execution we have significantly leveraged the NVIDIA CUDA

Asynchronous API [12] to best overlap PCIe transfers and MPI communication with GPU and CPU computation. Using knowledge of the task-graph and GPU task dependencies, the Uintah task scheduler is able to pre-fetch data needed for simulation variables prior to task execution [25]. When a GPU task has had its MPI communications requirements met and is ready to run, data needed for the task is resident in GPU main memory. The GPU task simply queries the GPU data warehouse for device pointers and invokes the computational kernel. The complexities and details of all device memory management and asynchronous operations are handled automatically by the Uintah infrastructure.

GPU tasks are assigned in a round-robin fashion to an arbitrary number of GPUs on-node once their asynchronous host-to-device data copies have completed. This design has made it possible to overlap MPI communication and asynchronous GPU data transfers with CPU and GPU task execution, significantly reducing MPI wait times [25]. Using the multiple copy engines available on NVIDIA Fermi and Kepler GPUs, GPU tasks can simultaneously copy data to-and-from the device as well as run multiple kernels. To facilitate this level of concurrency the Uintah task scheduler maintains a set of queues for *CUDA stream* and *event* handles and assigns them to each simulation variable to overlap with other host-to-device memory copies as well as kernel execution. In the presence of multiple on-node GPUs (e.g. NSF's Keeneland), the Uintah task scheduler also manages a CUDA calling context for each device. The task scheduler has a novel mechanism for detecting completion of asynchronous operations by querying GPU tasks waiting in the device work queues with CUDA API functions such as `cudaStreamQuery` on each of a task's recorded *CUDA events*.

This architecture has now also been used successfully in a prototype form with Intel Xeon Phi co-processors on NSF's Stampede, [40]. While the Uintah DAG-based approach performs well in many situations, its dynamic and asynchronous manner makes it difficult to analyze its performance as the necessary theoretical background is somewhat sparse [55]. The challenge that exists is then to understand exactly how Uintah is able to achieve scalability by examining how it modifies execution of its tasks in the light of varying processor and communications performance. It is the obtaining of this understanding that is the main task of this paper. This requires the different architectures and applications to be summarized as is done in the following section.

3. TARGET ARCHITECTURES

The three machines considered here illustrate some of the architectural differences in processor and network performance that may give rise to portability challenges. The main architectural features of these machines are summarized in Table 1.

3.1 TITAN

Titan is currently the fastest machine of the Top 500 list from November 2012 [62] with a theoretical peak of 27 petaflops and ranked third on the Green 500 list for energy efficiency. Each heterogeneous Cray XK7 node is equipped with a 16-core AMD Opteron 6274 processor running at 2.2 GHz, 32 gigabytes of DDR3 memory and a single NVIDIA Tesla K20 GPU with 6 GB GDDR5 ECC memory. The entire machine offers 299,008 CPU cores and 18,688 GPUs (1 per node) and over 710 TB of RAM. Even though the GPUs have a slower clock speed (732 MHz versus 2.2 GHz) than the CPUs, each GPU contains 2,688 CUDA cores. The overall system design was to use the CPU cores to allocate tasks to the GPUs rather than directly processing the data as in conventional supercomputers. Titan's Cray Gemini network is a 3D Torus with a latency of about 1.4 μseconds, with a peak of over 20 GB/second of injection bandwidth per node and a memory bandwidth of up to 52 GB/second per node.

From a software development perspective, Titan offers perhaps the greatest portability challenge for computational scientists to efficiently run on both host CPU and GPU. The Uintah Framework has been extended to incorporate a new task scheduler which offers three different ways of scheduling tasks to include MPI, threads and GPUs. To harness the computational power of the GPU, a new GPU kernel must be developed for each task. However, the Uintah framework infrastructure provides a seamless way of using the CPUs to allocate tasks and move data back and forth to the GPU without explicit calls from the computational stack.

3.2 Stampede

The NSF Stampede machine is ranked seventh in the top 500 [62] and provides an interesting alternative to Titan as it contains Intel's new co-processor technology, the Xeon Phi. Each of the 6400 Stampede nodes has two eight core Xeon E5-2680 operating at 2.7GHz with a 61 core Intel Xeon Phi co-processor with cores operating at 1.0GHz. The system interconnect is via a fat-tree FDR InfiniBand interconnect, [10], with a bandwidth of 56GB/second.

Stampede provides five programming models: Host-only, MIC native, offload, reverse offload and symmetric. Uintah currently only uses the host-only, native and symmetric models, as these were the fastest and most portable options, as described in [40]. A Uintah MIC offload model scheduler is under development.

In the host-only model, programs run only on host CPUs in the system without any utilization of the Xeon Phi co-processors. Host processors between multiple nodes can communicate though MPI. This model is similar to running on most other CPU-only clusters. For the symmetric model, programs can run on both the host CPU and the Xeon Phi co-processor card natively. MPI messages can be processed by host CPU and Xeon Phi directly. Minor modifications were made to the Uintah Framework to incorporate the hyrid scheduler which can schedule tasks on Xeon Phi in the same manner as for the host. The advantage of the Xeon Phi co-processor over the Titan GPU is that computational tasks running on the Xeon Phi do not have to be developed from scratch which is what is required for tasks running on Titan's GPUs.

3.3 MIRA

The Mira system is a new DOE open science 10-petaflop IBM Blue Gene/Q system installed at Argonne National Laboratory. Mira is designed to solve large-scale problems and has low power requirements and was ranked the fourth fastest supercomputer in the world as recorded by top500.org in November 2012 [62]. Each Mira compute node is equipped with 16 PowerPC A2 core processors running at 1.6GHz and 16 GB of SDRAM-DDR3 memory. The simple, low-power Blue Gene /Q cores support 4 hardware threads per core. With in-order execution, there is no instruction-level parallelism, so the key goal is to use multiple threads per core to maximize instruction throughput. The challenge in using Mira is that the lower floating point performance of the cores requires the use of a larger core count which in turn has the potential to stress a communications network that is very different from those of the other two machines considered here. [16], the Argonne Leadership Computing Facility's test and development rack of Blue Gene/Q. and are using gcc over gcc.legacy to utilize the fine-grained locking in MPICH as Uintah relies on MPI_THREAD_MULTIPLE support. [16], the Argonne Leadership Computing Facility's test and development rack of Blue Gene/Q. With the Blue Gene/Q software stack including standard runtime libraries for C, C++ and Fortran, Uintah was able to run on Mira without modification. Only minor

SYSTEM	Vendor / Type	CPU	Cores	Accel / Co-proc	Mem / node	Interconnect
Titan	Cray XK7	AMD Opteron 6200 @2.6GHz	299,008	Nvidia Tesla K20	32GB	Gemini
Stampede	Dell Zeus C8220z	Intel Xeon E5-2680 @2.7GHz Xeon Phi SE10P @1.1 GHz	102,400 $_{(host)}$ 390,400 $_{(phi)}$	Intel Xeon Phi	32GB	InfiniBand
Mira	IBM Blue Gene/Q	PowerPC A2 @1.6GHz	786,432	none	16GB	5D Torus

Table 1: System Specifications (compute nodes): Titan, Stampede and Mira

changes to the Uintah build system were necessary to recognize Blue Gene/Q-specific installation locations for MPI and compiler wrappers.

As Mira's power PC cores run at only 1.6GHz with a much simplified instruction set as compared to the Intel processors or AMD processors as used in Titan or Stampede, there are substantial differences in performance. The communications network on Mira is an integrated 5D torus with hardware assistance for collective and barrier functions and 2GB/sec bandwidth on all 10 links per node. The latency of the network varies between 80 nanoseconds and 3 microseconds at the farthest edges of the system. The interprocessor bandwidth per flop is close to 0.2, which is higher than many existing machines. This performance is offset by having only 1GB of memory per core, which can be problematic for certain applications, as we will see in Section 5.

Thus Mira has perhaps the best communications bandwidth per node relative to its computational power, and Stampede perhaps the (relatively) weakest of the three, while Titan is perhaps in between unless substantial use is made of its GPU cards.

4. SIMULATION COMPONENTS

The three representative problems cover a broad range of typical Uintah applications ranging from fluid-structure interaction to turbulent combustion. In order to add a deliberate contrast to these cases we have also included the modeling of radiation using ray-tracing. A massively parallel implementation of this last problem involves severe challenges in that there is a substantial amount of global communication.

4.1 MPMICE

Fluid-structure-interaction problems represent an important and computationally demanding class of problems that have been part of the landscape for which Uintah was originally conceived. Broadly speaking fluid-structure-interaction problems require the solution of the general multi-material computational fluid dynamics (CFD) formulation coupled to a solid mechanics computation. The MPMICE component uses the algorithm [21] to solve the governing multi-material formulation for the Navier-Stokes equations coupled to the MPM Lagrangian particle method for discretizing the solid mechanics. Additional sub-components are implemented such as various equations of state, constitutive models and solids→gas reaction models.

In the CFD, multi-material formulation, each material whether it is a fluid or a solid is defined at the continuum level over the entire computational domain even in regions where no material is present. The physical state of the system which includes mass, momentum and energy along with the volume fraction of each material is defined at every point in the computational grid. The volume fraction of all materials must sum to unity in any grid cell [29–31].

A cell centered finite volume solution technique [30] is used to solve for the discretized multi-material equations. The single control volume used for all materials simplifies the solution of the integral conservation equations of mass, momentum and energy and the exchange of quantities between the various materials. This method is fully compressible and has been extended to high speed flows [63] which are important in simulations involving explosions and in particular detonations which has been the subject of numerous projects for which Uintah was originally conceived.

The multi-material equations include exchange terms for mass, momentum, and heat. Mass exchange is based on solids→gas reaction models. Momentum and heat exchange is modeled by a drag law using the relative material velocities or temperatures.

The material point method (MPM) evolves the equations of motion of the solid and has been used in a variety of computational solid mechanics simulations [3, 57]. The material points represent the discretized solid volume and also represent the state of the system including mass, volume, velocity and stress. The particle state is interpolated to a background Cartesian grid which is the same grid used by the ICE multi-material CFD component. During each timestep, the state of the particles is mapped onto the computational grid. The states of all materials including both solid and fluid are computed. For the solid materials, changes in state computed on the background grid are interpolated back to the particles.

The use of the MPM Lagrangian particles to represent solids and the formulation of the CFD algorithm which does not explicitly differentiate between solids and fluids lends itself to a more efficient computational solution procedure where fluid-structure interfaces are not explicitly tracked nor are boundary conditions passed through the interfaces [29, 30]. The MPM Lagrangian particles are used to maintain the integrity of the fluid-solid interface while providing a method to track the deformation history of the solid(s).

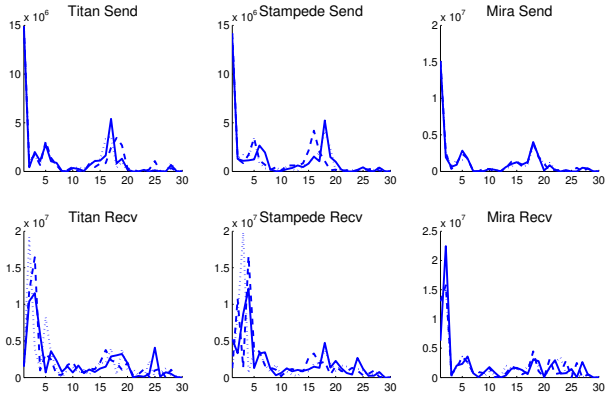

Figure 2: Send and Receive Volume Time Distribution Measurement for AMR MPMICE

The challenges with the scalability of this problem arise with the complex combination of solids, fluids and mesh refinement, see [42] for a full description of the changes to the runtime system needed in order to achieve scalability on this problem and others like it. Given the difficulty of achieving scalability it is by no means clear that it will be possible to achieve scalability on all three of the target machines.

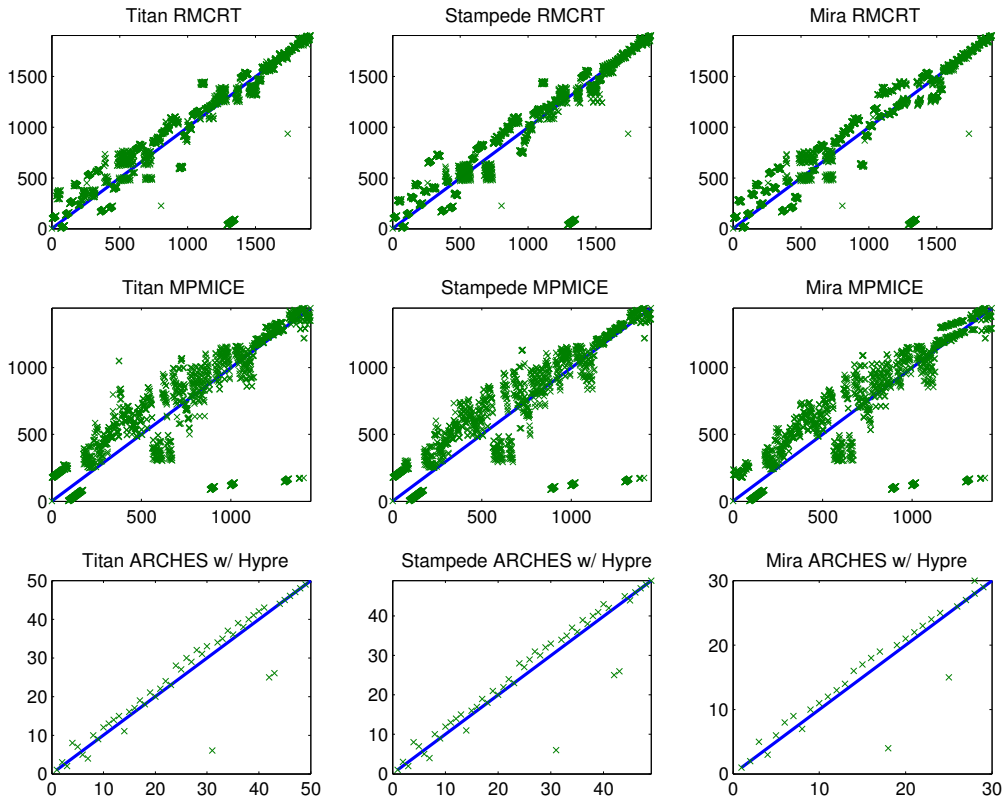

Figure 3: Task Execution Order for Each Simulation Component on Three Supercomputers (as relative to designed static algorithm)

Figure 2 shows the send and receive volume time distribution of communications for the case of three sample timesteps as three separate lines. The top three sub-figures show the data volumes sent over time by a Titan, Stampede and Mira node while the bottom three sub-figures show the data received by a Titan, Stampede and Mira node. In each timestep, the wall clock time is normalized and divided into 30 equal intervals. The MPI message volume for each interval is plotted as the y axis. The initial peak of data sent at the left of the top three sub-figures shows the transmission of the data that was computed at the end of the last timestep. For example, as soon as current timestep starts, all the ghost cell data from old data warehouse can be send out immediately. Most of this initial data sent involves local communication and can be posted immediately.

However, due to the network latency and bandwidth limitations, those messages continue to be received over several of the 30 time intervals shown. The relative delay in receiving these initial sends on Stampede is longer than on the other two machines. For Stampede, the send and receives use the Sandy Bridge nodes via IB.The top three sub-figures show several later peaks of data sent after the initial sends. All those later peaks are due to the requirement to send ghost-cell data on newly computed variables by later tasks in the same timestep. By observing the receive side, we can see the receiving delays for the later peaks vary both from machine to machine and from timestep to timestep. Those changes may be related to many possible situations, e.g. network bandwidth usage of other users in the same machine which is hard to predict by using traditional static analysis of DAG's critical path. Uintah uses dynamic task scheduling that can automatically pick ready tasks to overlap those unpredictable communication delays.

This overlapping is done by the Uintah task scheduler moving later tasks to execute earlier than would otherwise be the case.

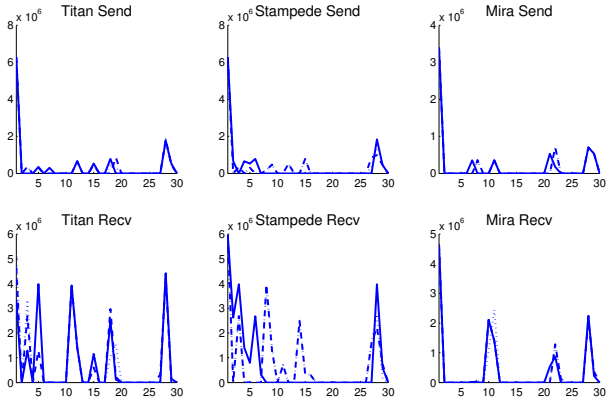

Figure 4: Send and Receive Volume Time Distribution Measurement for ARCHES with Hypre

Figure 3 shows the real scheduling task order in the y direction and its ordinal designed task order in the x direction. The solid line represents how tasks will be executed when static scheduling is used. The scatter points (x) show the task execution order when the multi-queue scheduler is used. All tasks below the solid diagonal line are executed earlier than they would be if static execution is used. All tasks above the solid diagonal line are executed later than they would have been if static execution had been used.

The MPMICE scheduling results show that many tasks are moved to a region close to x-axis. For example, some task originally designed to be execute as in between the $1300th$ and $1400th$ tasks are moved to approximately the first 100 tasks to be run. In this way, the time that would be spent waiting for initial MPI sends to

arrive is hidden. We also see that for Stampede, more tasks are moved to be executed earlier than on Titan and Mira, for example see tasks numbered from $500 - 700$ for static execution. This observation matches the comment that some messages are delayed in Stampede, so the scheduler can move tasks to overlap this delay accordingly. Overall this shows how the task execution order depends on a combination of the core clock speeds and the communications performance.

4.2 ARCHES

The ARCHES component is a multi-physics Large Eddy Simulation (LES) numerical algorithm used to solve for the reacting flow field and heat transfer arising in a number of computationally demanding simulations such as oxy-coal fired boiler simulations, oil recovery, and complicated pipe mixing. In addition to solving the multi-material Navier-Stokes equations, sub-grid models are used to describe the complicated turbulence of momentum and species transport.

The ARCHES component solves the coupled Navier-Stokes equations for mass, momentum and energy conservation equations. The algorithm uses a staggered finite-volume mesh for gas and solid phase combustion applications [17, 47, 48]. The discretized equations are integrated in time using an explicit strong-stability preserving third-order Runge-Kutta method [20]. Spatial discretization is handled with central differencing where appropriate for energy conservation or flux limiters (eg, scalar mixture fractions) to maintain realizability. In contrast to the explicit formulation of ICE, ARCHES uses the low-mach, pressure formulation which requires a solution of an implicit pressure projection at every timestep using the Hypre linear solver package [18].

For momentum and species transport equations, a dynamic, large eddy turbulence closure model is used to account for sub-grid velocity and species fluctuations [49]. The gas phase chemistry for coal combustion is represented using a chemical equilibrium model parameterized by the coal gas and secondary oxidizer stream mixture fractions [54]. The energy balance includes the effect of radiative heat-loss/gains in the IR spectra by solving the radiative intensity equation using a discrete-ordinance solver [33]. The solution procedure solves the intensity equation over a discrete set of ordinances which is formulated as a linear system similar to the pressure projection equation and is solved using Hypre [18]. The gas phase chemistry is parameterized by the two mixture fractions and heat-loss terms and preprocessed in a tabular form for dynamic table look-up during the course of the LES simulation.

The challenging nature of this problem lies in the complex physics and variety of numerical techniques used. Good scalability has been achieved with the Taylor Green Vortex Problem described in [51]. This involved a careful use of both data structures and options in the Hypre code so it is far from clear that the same scalability will transfer from Titan to the other two machines.

Figure 4 also shows the send and receive volume time distribution as the similar way to that of MPMICE. Again the initial peak of the data sent also distributes the data that is computed from the last timestep and these messages can be send out immediately. Unlike MPMICE, ARCHES is not continually receiving data. The reason is that the ARCHES component calls Hypre to do a number of iterative linear solves. During those linear solution phases, the MPI communicator is passed to the Hypre library. In particular, when Hypre is running we can see there are several intervals when there are no receives seen by ARCHES, as Uintah is not able trace any communications conducted within the Hypre library. This effectively means that data cannot be passed to a Uintah task unless all processors have finished that specific Hypre task. MPI_Irecv for all tasks after this Hypre solve (essentially a global synchronization) can only be posted after Hypre has finished, thus resulting in several peaks in the the data received.

Although Hypre now supports the use of a hybrid OpenMP/MPI parallel approach, we have so far been unable to make use of this in a way that is consistent with the Uintah multi-threaded approach on a multicore node and so used our MPI only scheduler for this problem. Therefore, there are far less tasks per MPI node for ARCHES as we can see from its task execution order plots in Figure 3. The multiple global synchronization points further limited the task scheduler's ability to migrate Uintah tasks, as the scheduler only allows tasks to be moved before a global synchronization point, and not after to avoid scheduling deadlock. However, we can still see that some tasks are moved to be executed early to overlap the initial sends.

4.3 Reverse Monte Carlo Ray Tracing (RMCRT)

Scalable modeling of radiation is important in a number of multiple applications areas such as heat transfer in combustion [53], neutron transport modeling [11] in nuclear reactors and astrophysics modeling [65] and is currently one of the most challenging problems in present-day large-scale computational science and engineering, due to the global nature of radiation. An approach using Reverse Monte Carlo Ray Tracing (RMCRT) is used here for radiation modeling in which rays are followed back from the source to the various origins. While this is efficient in that it does not follow rays that do not reach the source, the computational and communications complexity is still potentially prohibitive on a modern heterogenous machine even though rays may be traced independently on a single GPU accelerator at high speed.

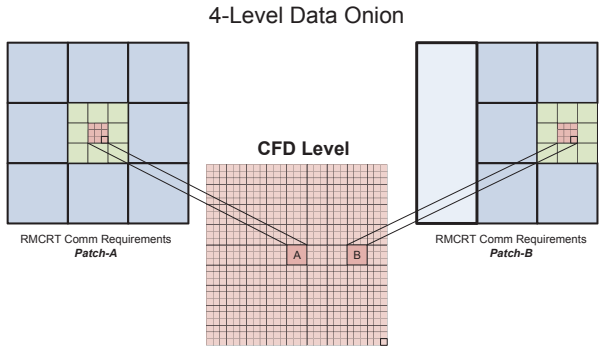

Figure 5: RMCRT Mesh Coarsening Scheme

The solution often adopted and used here is that multiple length scales are used to ensure that the amount of communication and computation is reduced in a way that is consistent with achieving accuracy. This may be illustrated by Figure 5. This figure shows a dense computational fluids mesh and for two of the computational fluids cells shows the radiation mesh. The radiation mesh may be coarsened rapidly away from the particular cell that the (reverse) rays originate in. The number of rays is kept constant per cell, as are the heat fluxes that are calculated.

The approach used by [25, 26, 58, 59] is to store the entire geometry on each single multicore node and to calculate the partial heat fluxes on the boundaries due to the radiation originating locally. Suppose that there are N_{total} fine mesh cells. The algorithm then involves a broadcast of all the data associated with every cell to every other cell on all the other processors. This involves a multiple of N_{total}^2 in terms of total communication. The volume of communication in this case may overwhelm the system for large

problems in our experience. The alternative is to use coarser resolutions as shown in Figure 5 in which a fine mesh is only used close to each point and a coarse mesh used further away. The use of adaptive meshes in the context of radiation is well understood with more traditional approaches [11, 50, 65], such as the Discrete Ordinates (DO) method used in the ARCHES combustion component of the Uintah code, [27]. However, the DO approach as used in Section 4.2 with ARCHES is costly and may consume as much as 60-70% of the calculation time. In applications where such high accuracy is important, RMCRT can become more efficient than DO approaches. In particular, RMCRT can potentially reduce the cost on shared memory machines [24, 58, 59] and on distributed memory [25, 26], with GPU accelerators [39]. A simple analysis of the two level scheme of [26] breaks the method down into the following steps:

1. Replicate the geometry (once) and coarsen mesh solution of temperature and absorption coefficients (every timestep) on all the nodes using allgather; This has a complexity of $\alpha log(p) + \beta \frac{p-1}{p}(N/r)^3$ for p cores with N^3 elements per mesh patch on a core are coarsened by a factor of r, where α is the latency and β the transmission cost per element [61].

2. Carry out the computationally very intensive ray-tracing operation locally. Suppose that we have r_a rays per cell then each ray has to be followed through as many as λN_G coarse mesh cells, where $N_G \approx Np/r$, or a multiple of this if there is reflection and where $0 \leq \lambda \leq \sqrt{3}N$. The total work is thus the sum of the fine mesh on each node contribution and the contribution from all the coarse mesh cells: $(\lambda N^4 + \lambda N_G^4)W_{ray}$, where W_{ray} is the work per ray per cell.

3. Distribute the resulting divergences of heat fluxes back to all the other nodes, again this cost is $\alpha log(p) + \beta \frac{p-1}{p}(N/r)^3$.

The relative costs of computation vs. communication are then given:

$$\lambda N^4(1 + (p/r)^4)W_{ray}r_a \;\; vs \;\; 2(\alpha log(p) + \beta \frac{p-1}{p}(N/r)^3)$$

Thus for enough rays r_a with enough refinement by a factor of r on the coarse radiation mesh, it looks likely that computation will dominate. A key challenge is that storage of $O(N_G^3)$ will be required on a multicore node and so only coarse and AMR mesh representations will be possible in a final production code at very large core counts. Although preliminary, this analysis can be extended to fully adaptive meshes, rather than the two level case considered here.

The send and receive volume time distribution for RMCRT shown in Figure 6, indicates that the volume of initial data sent dominates. As we discussed above, RMCRT is a relatively simple algorithm in that most of requisite data is only required from the last timetep and can be sent out once the current timestep starts. However, unlike the AMR MPMICE component for which most messages are local, RMCRT requires an all-to-all data transfer and the data transfer time is limited by the global bandwidth. From the bottom of Figure 6, it is seen that some MPI messages on Titan are delayed. We can also see large periods of time in which no data is being read, as RMCRT is computationally expensive and has a relatively simple variable exchange data pattern when compared to the complexity and frequency of communications in the multi-physics AMR MPMICE component.

Figure 3 shows that in order to overlap the huge initial send of data, the scheduler moved tasks as much as possible to be executed early (close to the x-axis). In other words some tasks that should

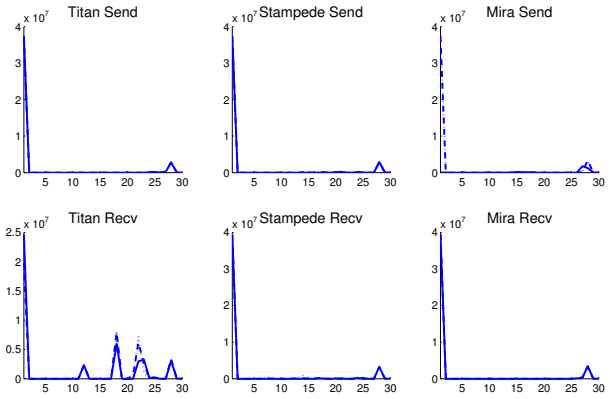

Figure 6: Send and Receive Volume Time Distribution Measurement for RMCRT

be statically scheduled for execution near the end of a timestep are moved to the beginning of the timestep. Furthermore, the MPI message delay on Titan is addressed by the scheduler moving tasks more aggressively than for Mira and Stampede.

5. SCALING RESULTS

In this section, we will show the scalability results for AMR MPMICE, ARCHES and RMCRT on the three target machines described in Section 3; Titan, Stampede and Mira. We also include preliminary GPU scaling results for RMCRT here. Co-processor scaling results are covered in [40]. We define strong scaling as a decrease in execution time when a fixed size problem is solved on more cores, while weak scaling should result in constant execution time when more cores are used to solve a correspondingly larger problem.

5.1 Strong Scaling of MPMICE

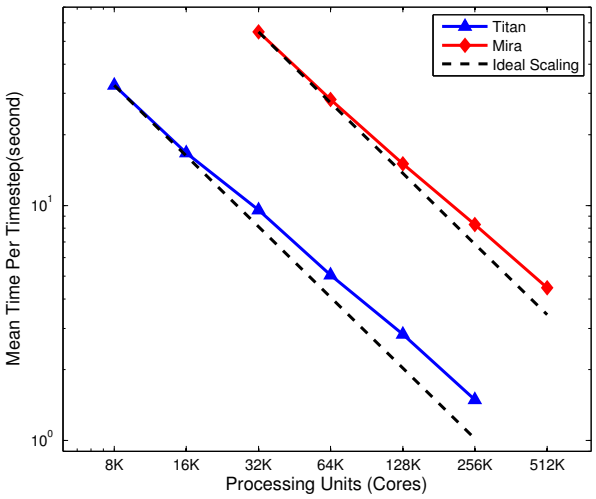

Figure 7: AMR MPMICE Strong Scaling

Perhaps the most satisfactory result from this study is shown in Figure 7 which displays the strong scaling results from the AMR MPMICE problem. This problem [42] exercises all of the main features of AMR, ICE and MPM and also includes a model for the deflagration of the explosive and the material damage model

ViscoSCRAM. The simulation grid utilized three refinement levels with each level being a factor of four more refined than the previous level. A total of 3.62 billion particles and 160 million cells are created on three AMR levels. These tests were run on Titan and Mira with up to 512K cores and with 16 threads per MPI node. Stampede results for MIC Native and Symmetric modes are not shown here but can be found in [40].The strong scaling efficiency is 68% on Titan 256K cores and 76% on Mira 512K core. Although not shown in Figure 7 or 8, we were able to successfully run the AMR MPMICE problem to completion at the full machine capacity of Mira (768k cores), though not with the same configuration as in Figure 7.

5.2 Weak Scaling of MPMICE and ARCHES

Figure 8 shows the weak scaling results from the AMR MPMICE problem. This problem is the same type of problem with three refinement levels shown previously for strong scaling, but with different resolutions on each run. The average numbers of particles and cells per node are set as close as possible. In the same number of cores group, from left to right are results from Mira, Titan and Stampede, denoted by M, T and S respectively. The times for task execution, message passing and packing are shown in different colors. The weak scaling for all three machines for this example is almost perfect with higher than 95% efficiency.

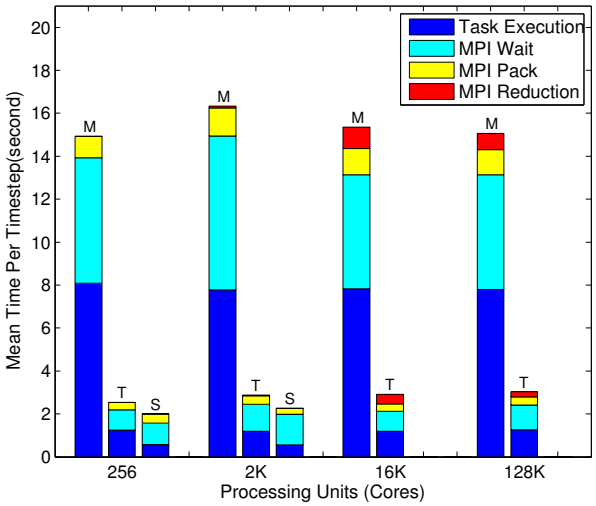

Figure 8: AMR MPMICE Weak Scaling

The weak scalability of the ARCHES component is shown in Figure 9 with each core group from left to right representing the results for Mira, Titan and Stampede, denoted by M, T and S respectively. For Mira and Titan, core counts ranged from 2K to 128K, and for Stampede, the core counts ranged from 2K to 64K. The problem uses a fixed resolution (42^3 cells per patch) with a single patch per core. Each timestep was broken down into linear solver time (Hypre Time) and Uintah time. For each timestep, the solution to a large (74,000 unknowns per core), sparse system of equations (Pressure-Poisson equation) is solved. The Hypre solver parameters used included the following: conjugate gradient method with the PFMG multi-grid preconditioner and a red-black Gauss-Seidel relaxer. The weak scaling efficiency is 88% on Mira at 128K cores, 79% on Titan at 128K cores and 56% on Stampede at 64K cores. However we can see that in most cases, the Uintah scaling component is better than Hypre component. The efficiency losses come mostly from the Hypre solving phase which has a log(P) term,

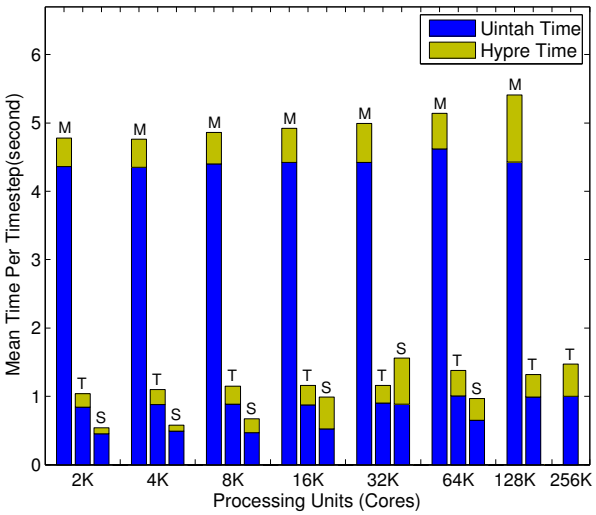

Figure 9: ARCHES with Hypre Weak Scaling

where P is the number of cores [51]. The relatively slower Mira cores (compared to Titan and Stampede) contributes to the significantly slower mean time/timestep. However, the network topology of Mira is well balanced and shows excellent weak scalability out to 128K cores. In contrast to Mira, the faster cores of Stampede and Titan magnify any slight timing variabilities as core counts were increased. The variabilities for Stampede were especially pronounced for 16K and 32K cores. The variability in timing may be due in part to system loads and network traffic impacting the Uintah simulation. We encountered issues with Mira when attempting to scale beyond 128K cores with an out of memory condition, which may be due to the limited memory per core.

5.3 Strong Scaling of RMCRT

Although solving the radiative transport equation using methods such as the parallel Discrete Ordinates Method (DOM) has been shown to scale [52] through the use of Hypre [18], the cost is significant due to large number of systems of linear equations required by this method. RMCRT has been shown to significantly reduce this cost. However, as mentioned in Section 4, RMCRT is an all-to-all method, where all geometry information and property model information for the entire computational domain must be present on each processor. This presently limits the size of the problem that can be computed due to memory constraints.

For the RMCRT problem, a two level coarsening scheme with a refinement ratio of 2 was used. The finer, CFD level used a resolution of 256^3 with one patch per core in each case and the coarse, RMCRT level used a resolution of 64^3 with a single patch. 10 rays per cell were used in the RMCRT portion of the calculation and the mean time per timestep was was averaged over 10 timesteps. Figure 10 shows strong scaling results from the RMCRT benchmark case detailed in Section 4 to 16k cores. These results are significant in that other adaptive mesh codes using similar approaches in radiative shock hydrodynamics, astrophysics and cosmology, e.g. Crash and Enzo [64, 65] report scaling to a maximum of near 1000 cores. The hybrid memory approach used by Uintah has also contributed to our results, as only one copy of the geometry is needed per multi-core node.

The results in Figure 10 also show that in this case Titan outperforms Stampede (host native mode) and Mira. In the case of the AMR MPMICE problem, Stampede and Titan perform similarly.

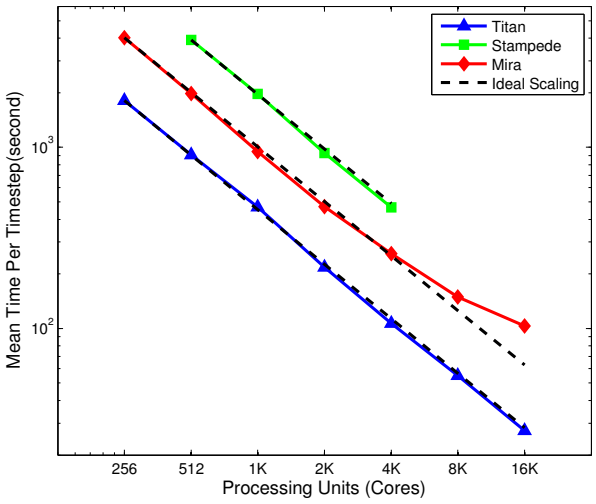

Figure 10: RMCRT Strong Scaling

The difference in these two cases would appear to be better network performance of Titan for the very large amounts of all-to-all communication required by the RMCRT problem.

Strong scaling results for a single-level, GPU-enabled RMCRT problem are shown by [25] for a prototype Uintah testbed component with a resolution of 128^3 on the TitanDev system. In moving to the full Titan system with its new NVIDIA K20 GPUs we found scaling results consistent to those in [25]. The only significant difference was that the Kepler GPU's were nearly three times faster than their Fermi predecessor as is shown in Figure 11. Although

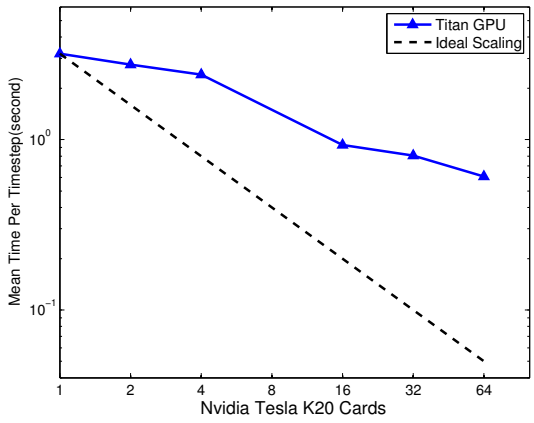

Figure 11: Single Level GPU-enabled RMCRT Strong Scaling

we were able to run successfully on Titan beyond 256 nodes, utilizing the on-node GPUs, the GPU implementation quickly runs out of work and strong scaling begins breaking down. The all-to-all nature of this problem severely limits the size of the problem that can be computed, and hence does not scale well due to memory constraints involved with large highly resolved physical domains [25]. To address this scalability issue and as part of future work, we will modify our RMCRT GPU implementation to leverage the multi-level coarsening scheme discussed in Section 4.

6. RELATED WORK

There are a number of computational frameworks that solver simlilar problems to Uintah. For example Flash [7, 50], p4est [9], ENZO [44] CASTRO [1] and Cactus [19]. Many of these codes tackle broad problem classes and achieve good computational results. Similarly, there are many codes that use a task-graph approach and significant corresponding analysis of this approach in areas such as linear equations solvers and indeed many other applications see, for example, [13, 23, 34] and numerable other references on this topic. What distinguishes Uintah from these efforts is the use of very large scale parallelism and the use of these techniques in a very general purpose computational PDEs framework for challenging engineering applications. What is also distinctive about the Uintah approach is that it combines a broad problems class with the use of the applications/runtine system to achieve scalability at very large core counts on challenging engineering problems. The approach that is perhaps closest to Uintah and indeed predates it, is that of Charm++ [8, 28] but as was remarked earlier although the task-graph approaches are conceptually similar, the architectures,implementations and problem classes solved are very different.

7. CONCLUSIONS AND FUTURE WORK

We have shown that using directed acyclic graphs to represent computational tasks combined with an asynchronous and dynamic runtime system provides an effective way to achieve scalable systems on disparate heterogenous and homogenous high performance computer systems. Porting Uintah to new heterogenous systems only requires changes to the runtime system such as schedulers and data warehouse, with little to no changes to infrastructure code. Porting Uintah task code can vary depending on the nature of the heterogeneous platform, however, the Uintah runtime system does provide convenient mechanisms to port any subset of task code. We have also shown that weak and strong scalability is achieved when the runtime environment is allowed to flexibly schedule the execution order of computational tasks to overlap computation with communication. It is the combination of the DAG representation of tasks with a runtime environment that can schedule tasks based on a pre-compiled dependency analysis in a preferred order that yields scalability on a variety of problems with widely different communication requirements, and on systems with very different architectures. A major remaining challenge is to extend Uintah to move beyond being able to use accelerators and co-processors to achieve scalability across the whole of machines like Titan, Mira and Stampede for very broad problem classes including components such as radiation.

8. ACKNOWLEDGMENTS

This work was supported by the National Science Foundation under subcontracts No. OCI0721659, the NSF OCI PetaApps program, through award OCI 0905068 and by DOE INCITE awards CMB026 and ENP009 for time on Titan and DOE NETL for funding under NET DE-EE0004449. Uintah was originally written under the Department of Energy, subcontract No. B524196. This research used the Argonne Leadership Computing Facility at Argonne National Laboratory, as supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. For early access to the TACC Stampede system we thank Karl Schulz, Bill Barth, Andy Terrel and all of TACC. For early access to ALCF's Mira and Vesta we thank Richard Coffey, Kalyan Kumaran and all of ALCF. We would also like to thank all those involved with Uintah, Steve Parker, Justin Luitjens, Todd Harman and Joseph Peterson in particular.

9. REFERENCES

[1] A Almgren, J Bell, D Kasen, M Lijewski, A Nonaka, P Nugent, C Rendleman, R Thomas, and M Zingale. Maestro, castro, and sedona–petascale codes for astrophysical applications. *arXiv preprint arXiv:1008.2801*, 2010.

[2] Saman Amarasinghe, Dan Campbell, William Carlson, Andrew Chien, William Dally, Elmootazbellah Elnohazy, Mary Hall, Robert Harrison, William Harrod, Kerry Hill, et al. Exascale software study: Software challenges in extreme scale systems. *DARPA IPTO, Air Force Research Labs, Tech. Rep*, 2009.

[3] S.G. Bardenhagen, J.E. Guilkey, K.M. Roessig, J.U. Brackbill, W.M. Witzel, and J.C. Foster. An Improved Contact Algorithm for the Material Point Method and Application to Stress Propagation in Granular Material. *Computer Modeling in Engineering and Sciences*, 2:509–522, 2001.

[4] M. Berzins. Status of Release of the Uintah Computational Framework. Technical Report UUSCI-2012-001, Scientific Computing and Imaging Institute, 2012.

[5] M. Berzins, J. Luitjens, Q. Meng, T. Harman, C.A. Wight, and J.R. Peterson. Uintah - a scalable framework for hazard analysis. In *TG '10: Proc. of 2010 TeraGrid Conference*, New York, NY, USA, 2010. ACM.

[6] M. Berzins, Q. Meng, J. Schmidt, and J. Sutherland. Dag-based software frameworks for pdes. In *Proceedings of HPSS 2011 (Europar, Bordeaux August, 2011)*, 2012.

[7] B.Fryxell, K.Olson, P.Ricker, F.X.Timmes, M.Zingale, D.Q.Lamb, P.Macneice, R.Rosner, J.W. Rosner, J.W. Truran, and H.Tufo. FLASH an adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes. *The Astrophysical Journal Supplement Series*, 131:273–334, November 2000.

[8] A. Bhatelé, L. V. Kalé, and S. Kumar. Dynamic topology aware load balancing algorithms for molecular dynamics applications. In *Proc. of the 23rd international conference on Supercomputing*, pages 110–116. ACM, 2009.

[9] C. Burstedde, L. C. Wilcox, and O. Ghattas. `p4est`: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM Journal on Scientific Computing*, 33(3):1103–1133, 2011.

[10] Texas Advanced Computing Center. Stampede Web Page, 2013. http://www.tacc.utexas.edu/resources/hpc/stampede.

[11] C.J. Clouse. Parallel deterministic neutron transport with amr. In Frank Graziani, editor, *Computational Methods in Transport*, volume 48 of *Lecture Notes in Computational Science and Engineering*, pages 499–512. Springer Berlin Heidelberg, 2006.

[12] Nvidia Corp. Nvidia Developer Zone Web Page, 2012. http://developer.nvidia.com/nvidia-gpu-computing-documentation.

[13] Mohammad I. Daoud and Nawwaf Kharma. A hybrid heuristic-genetic algorithm for task scheduling in heterogeneous processor networks. *J. Parallel Distrib. Comput.*, 71(11):1518–1531, November 2011.

[14] J. D. de St. Germain, J. McCorquodale, S. G. Parker, and C. R. Johnson. Uintah: A massively parallel problem solving environment. In *Ninth IEEE International Symposium on High Performance and Distributed Computing*, pages 33–41. IEEE, Piscataway, NJ, nov. 2000.

[15] Argonne Leadership Computing Facility. Mira Web Page, 2013. https://www.alcf.anl.gov/mira.

[16] Argonne Leadership Computing Facility. Vesta Web Page, 2013. https://www.alcf.anl.gov/vesta.

[17] M. Faghri and S.D.i.H.T.B Senden, editors. *Heat Transfer to Objects in Pool Fires*, volume 20. Wit Press, Southampton, UK, 2008.

[18] R.D. Falgout, J.E. Jones, and U.M. Yang. *Numerical Solution of Partial Differential Equations on Parallel Computers*, volume UCRL-JRNL-205459, chapter The Design and Implementation of Hypre, a Library of Parallel High Performance Preconditioners, pages 267–294. Springer-Verlag, 51, 2006.

[19] T. Goodale, G.Allen, G.Lanfermann, J.Masso, T. Radke, E.Seidel, and J.Shalf. The Cactus framework and toolkit: Design and applications. In *Vector and Parallel Processing VECPAR 2002*, Lecture Notes in Computer Science, Berlin, 2003. Springer.

[20] S. Gottlieb, C.W. Shu, and W. Tadmor. Strong stability-preserving high-order time discretization methods. *Siam Review*, 43(1):89–112, 2001.

[21] J. E. Guilkey, T. B. Harman, and B. Banerjee. An eulerian-lagrangian approach for simulating explosions of energetic devices. *Computers and Structures*, 85:660–674, 2007.

[22] J. E. Guilkey, T. B. Harman, A. Xia, B. A Kashiwa, and P. A. McMurtry. An Eulerian-Lagrangian approach for large deformation fluid-structure interaction problems, part 1: Algorithm development. In *Fluid Structure Interaction II*, Cadiz, Spain, 2003. WIT Press.

[23] Azzam Haidar, Hatem Ltaief, Asim YarKhan, and Jack Dongarra. Analysis of dynamically scheduled tile algorithms for dense linear algebra on multicore architectures. *Concurrency and Computation: Practice and Experience*, 24(3):305–321, 2011.

[24] J. R. Howell. The monte carlo in radiative heat transfer. *Journal of Heat Transfer*, 120(3):547–560, 1998.

[25] A. Humphrey, Q. Meng, M. Berzins, and T. Harman. Radiation Modeling Using the Uintah Heterogeneous CPU/GPU Runtime System. In *Proceedings of the 1st Conference of the Extreme Science and Engineering Discovery Environment (XSEDE 2012)*. ACM, 2012.

[26] I. Hunsaker, T. Harman, J. Thornock, and P.J. Smith. Efficient Parallelization of RMCRT for Large Scale LES Combustion Simulations. Paper AIAA-2011-3770. 41st AIAA Fluid Dynamics Conference and Exhibit, 2011.

[27] J.Spinti, J. Thornock, E. Eddings, P.J. Smith, and A. Sarofim. Heat transfer to objects in pool fires, in transport phenomena in fires. In *Transport Phenomena in Fires*, Southampton, U.K., 2008. WIT Press.

[28] L. V. Kale, E. Bohm, C. L. Mendes, T. Wilmarth, and G. Zheng. Programming petascale applications with Charm++ and AMPI. *Petascale Computing: Algorithms and Applications*, 1:421–441, 2007.

[29] B. A. Kashiwa. A multifield model and method for fluid-structure interaction dynamics. Technical Report LA-UR-01-1136, Los Alamos National Laboratory, 2001.

[30] B. A. Kashiwa and R. M. Rauenzahn. A cell-centered ICE method for multiphase flow simulations. Technical Report LA-UR-93-3922, Los Alamos National Laboratory, 1994.

[31] B.A. Kashiwa and E.S. Gaffney. Design basis for cfdlib. Technical Report LA-UR-03-1295, Los Alamos National

Laboratory, 2003.

[32] B.A. Kashiwa, M.L. Lewis, and T.L. Wilson. Fluid-structure interaction modeling. Technical Report LA-13111-PR, Los Alamos National Laboratory, 1996.

[33] G. Krishnamoorthy. *Predicting Radiative Heat Transfer in Parallel Computations of Combustion*. PhD thesis, University of Utah, Salt Lake City, UT, December 2005.

[34] Yu-Kwong Kwok and Ishfaq Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.*, 31(4):406–471, December 1999.

[35] J. Luitjens and M. Berzins. Improving the performance of Uintah: A large-scale adaptive meshing computational framework. In *Proc. of the 24th IEEE Int. Parallel and Distributed Processing Symposium (IPDPS10)*, 2010.

[36] J. Luitjens and M. Berzins. Scalable parallel regridding algorithms for block-structured adaptive mesh refinement. *Concurrency and Computation: Practice and Experience*, 23(13):1522–1537, 2011.

[37] J. Luitjens, M. Berzins, and T. Henderson. Parallel space-filling curve generation through sorting. *Concurr. Comput. : Pract. Exper.*, 19(10):1387–1402, 2007.

[38] J. Luitjens, B. Worthen, M. Berzins, and T. Henderson. *Petascale Computing Algorithms and Applications*, chapter Scalable parallel amr for the Uintah multiphysics code. Chapman and Hall/CRC, 2007.

[39] Q. Meng, A. Humphrey, and M. Berzins. The Uintah Framework: A Unified Heterogeneous Task Scheduling and Runtime System. In *Digital Proceedings of Supercomputing 12 - WOLFHPC Workshop*. IEEE, 2012.

[40] Q. Meng, A. Humphrey, J. Schmidt, and M. Berzins. Preliminary Experiences with the Uintah Framework on Intel Xeon Phi and Stampede. In *The 2nd Conference of the Extreme Science and Engineering Discovery Environment (XSEDE 2013)*. ACM, 2013.

[41] Q. Meng, J. Luitjens, and M. Berzins. Dynamic task scheduling for the uintah framework. In *Proceedings of the 3rd IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS10)*, 2010.

[42] Qingyu Meng and Martin Berzins. Scalable large-scale fluid-structure interaction solvers in the Uintah framework via hybrid task-based parallelism algorithms. *Concurrency and Computation: Practice and Experience*, 2013.

[43] U.S. Department of Energy Oak Ridge National Laboratory and Oak Ridge Leadership Computing Facility. Titan Web Page, 2013. http://www.olcf.ornl.gov/titan/.

[44] B. O'Shea, G. Bryan, J. Bordner, M. Norman, T. Abel, R. Harkness, and A. Kritsuk. Introducing Enzo, an amr cosmology application. In *Adaptive Mesh Refinement - Theory and Applications*, volume 41 of *Lecture Notes in Computational Science and Engineering*, pages 341–350, Berlin, Heidelberg, 2005. Springer-Verlag.

[45] S. G. Parker. A component-based architecture for parallel multi-physics PDE simulation. *Future Generation Comput. Sys.*, 22:204–216, 2006.

[46] S. G. Parker, J. Guilkey, and T. Harman. A component-based parallel infrastructure for the simulation of fluid-structure interaction. *Engineering with Computers*, 22:277–292, 2006.

[47] J. Pedel, J. Thornock, and P.J. Smith. Ignition of co-axial oxy-coal jet flames: data collaboration of experiments and simulations. *Combustion and Flame*, Accepted for publication, 2012.

[48] J. Pedel, J. Thornock, and P.J. Smith. Large simulation of pulverized coal jet flame ignition using the direct quadrature method of moments. *Energy and Fuels*, Accepted for publication, 2012.

[49] Stephen B. Pope. *Turbulent Flows*. Cambridge Press, 2000.

[50] E.-J. Rijkhorst, T. Plewa, A. Dubey, and G. Mellema. Hybrid characteristics: 3d radiative transfer for parallel adaptive mesh refinement hydrodynamics. *Astronomy and Astrophysics*, 452(3):907–920, 2006.

[51] J. Schmidt, M. Berzins, J. Thornock, T. Saad, and J. Sutherland. Large Scale Parallel Solution of Incompressible Flow Problems using Uintah and hypre. In *Proceedings of CCGrid 2013*. IEEE/ACM, 2013.

[52] J. Schmidt, J. Thornock, J. Sutherland, and M. Berzins. Large Scale Parallel Solution of Incompressible Flow Problems using Uintah and Hypre. Technical Report UUSCI-2012-002, Scientific Computing and Imaging Institute, 2012.

[53] P. J. Smith, R.Rawat, J. Spinti, S. Kumar, S. Borodai, and A. Violi. Large eddy simulation of accidental fires using massively parallel computers. In *18th AIAA Computational Fluid Dynamics Conference*, June 2003.

[54] L.D. Smoot and P.J. Smith. *Coal Combustion and Gasification*. Plenum Press, 1985.

[55] M. Snir. Private communication, 2013.

[56] D. Sulsky, Z. Chen, and H.L. Schreyer. A particle method for history-dependent materials. *Computer Methods in Applied Mechanics and Engineering*, 118(1-2):179–196, 1994.

[57] D. Sulsky, S. Zhou, and H. L. Schreyer. Application of a particle-in-cell method to solid mechanics. *Computer Physics Communications*, 87:236–252, 1995.

[58] X. Sun. *Reverse Monte Carlo ray-tracing for radiative heat transfer in combustion systems*. PhD thesis, Dept. of Chemical Engineering, University of Utah, 2009.

[59] Xiaojing Sun and Philip J. Smith. A parametric case study in radiative heat transfer using the reverse monte-carlo ray-tracing with full-spectrum k-distribution method. *Journal of Heat Transfer*, 132(2), 2010.

[60] Texas Advanced Computing Center. Stampede User Guide, 2013. http://www.tacc.utexas.edu/user-services/user-guides/stampede-user-guide.

[61] R. Thakur, R. Rabenseifner, and W. D. Gropp. Optimization of collective communication operations in mpich. *International Journal of High Performance Computing Applications*, 19(1):49–66, 2005.

[62] TOP500.Org. Top500 Web Page - November, 2012, 2012. http://www.top500.org/lists/2012/11/.

[63] L.T. Tran and M. Berzins. IMPICE Method for Compressible Flow Problems in Uintah. *International Journal For Numerical Methods In Fluids*, 2011.

[64] B. van der Holst, G. Toth, I.V. Sokolov, K.G. Powell, J.P. Holloway, et al. Crash: A Block-Adaptive-Mesh Code for Radiative Shock Hydrodynamics - Implementation and Verification. *Astrophys.J.Suppl.*, 194:23, 2011.

[65] John H. Wise and Tom Abel. enzo+moray: radiation hydrodynamics adaptive mesh refinement simulations with adaptive ray tracing. *Monthly Notices of the Royal Astronomical Society*, 414(4):3458–3491, 2011.